Disclaimer:

The audit makes no statements or warranties about the utility of the code, the safety of the code, the suitability of the business model, the regulatory regime for the business model, or any other statements about the fitness of the contracts to the purpose or their bug-free status. The audit documentation is for discussion purposes only.

Overview:

We conclude that the project was developed using Remix IDE, a good IDE for debugging and interacting directly with live networks and quick prototyping. However, alternative js ecosystems like hardhat are more suitable for this project.

The project doesn't have any test suit, which is a bad practice for developing a smart contract. Having close to 100% test coverage guarantees that your code works as intended and empowers us, auditors, to spend more time identifying security issues rather than functional Bugs.

Contract Audited: ERC1404TokenMinKYCv13.sol

Complete Analysis:

Low: Lack of zero address check. (Unresolved)

Recommendation: Check if `*account*` is not a zero address in mint and burn function.

Informational: Unnecessary packing. (Unresolved)

The EVM works with 256bit/32byte words. Every operation is based on these base units. If your data is smaller, further operations are needed to downscale from 256 bits to 64 bits.

Recommendation: Use uint256 data type instead of uint64 for values of _receiveRestriction & _sendRestriction mapping. Doing so will reduce 100 gas cost/ account.

Informational: Variables can be marked as constant. (Unresolved)

Variables can be changed to public constants, since they are not changed throughout the code.

Recommendation: Variables: `*version*`, `*IssuancePlatform*` and `*issuanceProtocol*`*could be marked as constant.*

Informational: Gas-efficient way could be used to throw. (Unresolved)

Using custom errors is cheaper in gas, especially in deployment.

Recommendation: All of the require statements could be changed to if statements that revert a custom Error.

Informational: Solidity version should be updated. (Unresolved)

Best practices for Solidity development requires strict and explicit usage of the latest stable version of Solidity, which is 0.8.17 at the moment.

Recommendation: Consider updating to “pragma solidity 0.8.17;”

Informational: Variable can be marked as immutable. (Unresolved)

Variable can be marked as immutable if it’s only being initialized during constructor initialization.

Recommendation: Variable: `*decimals*` *could be marked as* immutable*.*

Informational: Unnecessary calldata copy. (Unresolved)

Reading from calldata directly is cheap.

Recommendation: Use calldata data location instead of memory for string and address array params.

Informational: Misleading function name. (Unresolved)

Function is modifying/ setting KYC data whereas function name suggests otherwise.

Recommendation:Consider renaming bulkWhitelistWallets fn name.

Informational: Unnecessary internal function call. (Unresolved)

Recommendation: Instead of reading owner address via Ownable.owner() in the constructor function, make use of the global variable msg.sender; store msg.sender in memory var, and use it throughout the function, which will reduce significant gas fees.

Informational: Expensive read (Unresolved)

Reading from _msgSender() is expensive than global variable msg.sender

Recommendation: Consider using msg.sender instead of _msgSender()

Informational: Use optimal comparison operator (Unresolved)

When comparing uints, you can save gas costs by strategically picking the optimal operator. For example, it is cheaper to use strict < or > operators over <= or >= or != operators. This is due to the additional EQ opcode which must be performed.

Recommendation: Consider using *if(value < 1)* instead of *if(value <= 0*) in detectTransferRestriction function, also since the data type of *value* var is uint, so it would never be less than 0.